pyControl Apps #1 - BIG-IP Routing Table

Jason Rahm, 2009-15-07

Note: pycontrol v1 is deprecated. A new article and script have been added to work through the bigsuds library.

This week I will be continuing with my discovery into python and iControl... This time out I'll take a look at the BIG-IP routing table.

iControl SDK Pre-work

Looking into the online SDK for the routing table methods, it's clear that the only interface I need is Networking::RouteTable. The methods I'll use are:

- get_static_route (no parameters)
- get_static_route_type (parameter: routes, expects a list of routes)
    - For each route passed, returns a type:
        - ROUTE_TYPE_GATEWAY
        - ROUTE_TYPE_POOL
        - ROUTE_TYPE_INTERFACE
        - ROUTE_TYPE_REJECT
- get_static_route_gateway (parameter: routes, expects a list of routes)
- get_static_route_pool (parameter: routes, expects a list of routes)
- get_static_route_reject (parameter: routes, expects a list of routes)
- get_static_route_vlan (parameter: routes, expects a list of routes)
- get_management_route (no parameters)
- get_management_route_gateway (parameter: routes, expects a list of routes)

Note: The same methods exist for the management routes as do the static routes (gateway, pool, vlan, reject), but for this exercise I'll assume all management traffic is destined for an IP address.

pyControl Script Skelton - The Main Loop

As I've seen in most of Joe's iControl script efforts, as well as other places, its common to keep the main loop fairly light and shift most of the processing out to functions. To that end, I'll demonstrate the function approach and keep the main loop esentially a reusable skeleton. Only things that change signficantly here from script to script are the WSDL files in use and the functions called.

pyControl Main Loop

```
if __name__ == "__main__":
     import pycontrol.pyControl as pc
     import getpass
     from sys import argv
     if len(argv) != 3:
          exit("Usage: routeInfo.py  ")
     host = argv[1]
     uname = argv[2]
     print "%s, enter your " % getpass.getuser(),
     upass = getpass.getpass()
     b = pc.BIGIP(
          hostname = host,
          username   = uname,
          password   = upass,
          wsdl_files = ['Networking.RouteTable']
          )
     rt = b.Networking_RouteTable
     get_tmmRoutes(rt)
```

```
    get_mgmtRoutes(rt)
```

In the main loop, I've imported pyControl as pc, so when I call BIGIP, I can just use pc.BIGIP instead of pycontrol.pyControl.BIGIP. It's just a shortcut, either is acceptable. You'll note that I imported a couple python modules, getpass, and the argv function from the sys module. This is to handle username/password interaction and arguments, respectively. After collecting the hostname, username, & password, I define the BIGIP call and the wsdl files I want to load. Since I only need methods from Networking::RouteTable, there's no reason to load more. After creating another shortcut for the interface (rt for RouteTable), it's time to call some functions.

BIG-IP Routes

The BIG-IP supports the ability to route a network to an ip address, a pool, a vlan interface, or just reject it outright. Because of this, I need to query each route for the type of gateway that belongs to it. So once I've loaded all our routes into a variable with the get_static_route method call, I need to then iterate through the routes against their type with the get_static_route_type method call. Once that's done, I use the zip function to map the route to the gateway type for use later. This code performs these functions:

pyControl Method Calls

```
def get_tmmRoutes(obj):
    try:
        tmmStatic = obj.get_static_route()['return']
        tmmRtType = obj.get_static_route_type(routes = tmmStatic)['return']
    except:
        "Unable to fetch route information - check trace log"
    combined = zip(tmmStatic,  tmmRtType)
```

Now I need to create some empty lists for the routes before I iterate through each type. Because I want to sort the routes before printing them, I need to have a list of the routes to work with. This code inspects the type of route gathered from the methods calls, then utilizes the appropriate method call to append the gateway to the route, and finally, append the route+gateway dictionary to the list:

pyControl Method Calls

```
ldict_gw_ip = []
ldict_gw_pool = []
ldict_gw_vlan = []
ldict_gw_reject = []
for x in combined:
    if x[1] == 'ROUTE_TYPE_GATEWAY':
        x[0]['gateway'] = obj.get_static_route_gateway(routes = [x[0]])['return']
        ldict_gw_ip.append(x[0])
    if x[1] == 'ROUTE_TYPE_POOL':
        x[0]['gateway'] = obj.get_static_route_pool(routes = [x[0]])['return']
        ldict_gw_pool.append(x[0])
    if x[1] == 'ROUTE_TYPE_INTERFACE':
        x[0]['gateway'] = obj.get_static_route_vlan(routes = [x[0]])['return']
        ldict_gw_vlan.append(x[0])
    if x[1] == 'ROUTE_TYPE_REJECT':
        ldict_gw_reject.append(x[0])
```

Now that I have the lists of routes (in dictionary form with the lists), I need to sort the IP addresses. Using the built-in sort functions, it sorts .1, .10, .2, etc...so that's not going to work. I found a function to sort IP addresses in list format here on this Wad of Stuff blog, but I needed to pass a list of dictionaries, not a list of IP addresses. Unfortunately, my python-fu is still pretty green, but luckily, my posted question on said problem was met with an immediate response. So here's the functions for sorting a list of IP's, as well as a list of dictionaries where IP is a value in a dictionary. In the case of the iControl returned routes, it is a dictionary in the format **{'netmask': '255.255.255.0', 'destination': 192.168.1.0'}**, so I'll use the second function below. Note that you'll need to install the IPy module if you don't already have it.

Sort Function, List of IP's

```
def sort_ip_list(ip_list):
     from IPy import IP
     ipl = [ (IP(ip).int(),  ip) for ip in ip_list]
     ipl.sort()
     return [ip[1] for ip in ipl]
```

Sort Function, List of IP's within Dictionaries

```
def sort_ip_dict(ip_list):
     from IPy import IP
     ipl = [ (IP(i['destination']).int(),  i) for i in ip_list]
     ipl.sort()
     return [ip[1] for ip in ipl]
```

So now, I can actually send the lists to the sort function, then iterate through each list and print!

Sort and Print

```
gw_ip = sort_ip_dict(ldict_gw_ip)
gw_pool = sort_ip_dict(ldict_gw_pool)
gw_vlan = sort_ip_dict(ldict_gw_vlan)
gw_reject = sort_ip_dict(ldict_gw_reject)
print "\n"*2
print "TMM IP Routes: (net mask ip)"
for x in gw_ip:
     print "\t", x.get('destination') ,  x.get('netmask'),  x.get('gateway')[0]
print "\n"*2
print "TMM Pool Routes: (net mask pool)"
for x in gw_pool:
     print "\t", x.get('destination') ,  x.get('netmask'),  x.get('gateway')[0]
print "\n"*2
print "TMM Vlan Routes: (net mask vlan)"
for x in gw_vlan:
     print "\t",  x.get('destination') ,  x.get('netmask'),  x.get('gateway')[0]
print "\n"*2
print "TMM Rejected Routes: (net mask)"
for x in gw_reject:
     print "\t",  x.get('destination') ,  x.get('netmask')
```

And that's a wrap! Now that the code is complete (minus the management route function, but it's nearly identical), here's the output:

```
 Loading WSDL: Networking.RouteTable.wsdl

TMM IP Routes: (net mask ip)
     0.0.0.0 0.0.0.0 10.10.20.1
     172.18.1.0 255.255.255.0 10.10.20.1
     172.18.2.0 255.255.255.0 10.10.20.1
     172.18.10.0 255.255.255.0 10.10.20.1

TMM Pool Routes: (net mask pool)
     172.16.1.0 255.255.255.0 gateway_pool
     172.16.2.0 255.255.255.0 gateway_pool
     172.16.10.0 255.255.255.0 gateway_pool
```

```
TMM Vlan Routes: (net mask vlan)
        172.17.1.0 255.255.255.0 mgmt_tools
        172.17.2.0 255.255.255.0 mgmt_tools
        172.17.10.0 255.255.255.0 mgmt_tools

TMM Rejected Routes: (net mask)
        172.19.1.0 255.255.255.0
        172.19.2.0 255.255.255.0
        172.19.10.0 255.255.255.0

Management Routes: (net mask ip)
        0.0.0.0 0.0.0.0 192.168.1.254
```

For the complete code, navigate over to here in the wiki. Enjoy!